Evaluation of QOS-Control Servers on Real-Time Mach*

Kiyokuni Kawachiya[1], Masanobu Ogata[2], Nobuhiko Nishio[3], Hideyuki Tokuda[3]

[1] IBM Research, Tokyo Research Laboratory, <kawatiya@trl.ibm.co.jp>
[2] Power Mobile Systems, IBM Japan, Ltd.
[3] Faculty of Environmental Information, Keio University

Abstract. In an interactive multimedia environment that handles multiple sessions dynamically, a mechanism to control QOS among sessions is very important. As such a mechanism, the QOS-Control Server has been developed on RT-Mach. This paper gives the results of several experiments with this server, and describes the extensions of RT-Mach.

1 Introduction

In the Keio-MMP (MultiMedia Platform) project, we are extending Real-Time Mach 3.0 (RT-Mach) [1] and constructing servers for multimedia processing on our extension [2, 3]. One goal of our project is to incorporate the concept of Quality of Service (QOS), and to manage system resources over multiple continuous-media sessions on the basis of this concept. For such support, there should be some framework for QOS management at the system-software level. We adopted a QOS Manager-based scheme [4], and developed an experimental QOS-Control Server. This paper gives the results of several experiments with this server, and describes the extensions of RT-Mach needed for efficient QOS control.

There have been several studies of QOS-control architecture. The QoS-A by Lancaster University [5] tries to provide an integrated and coherent framework for QOS control. The IBM European Networking Center has proposed a media-scaling architecture with its transport system, HeiTS [6]. Most of these research projects focus on the QOS control in individual sessions over a network, while this paper focuses on dynamic QOS control among multiple sessions in a system.

2 A QOS-Control Server on RT-Mach

In RT-Mach, continuous-media processing programs can be written by using the system's periodic real-time thread, which processes a media data unit, such as a frame of video data, on every invocation. In this paper, such threads will be called "CM-Threads." Deadline misses of the thread can be reported through a deadline port. Using these features, we developed an experimental QOS-Control Server shown in Fig. 1(a). When a CM-Thread is started, it registers its "QOS

* This research is conducted under the Open Fundamental Software Technology Project of Information-technology Promotion Agency, Japan (IPA).

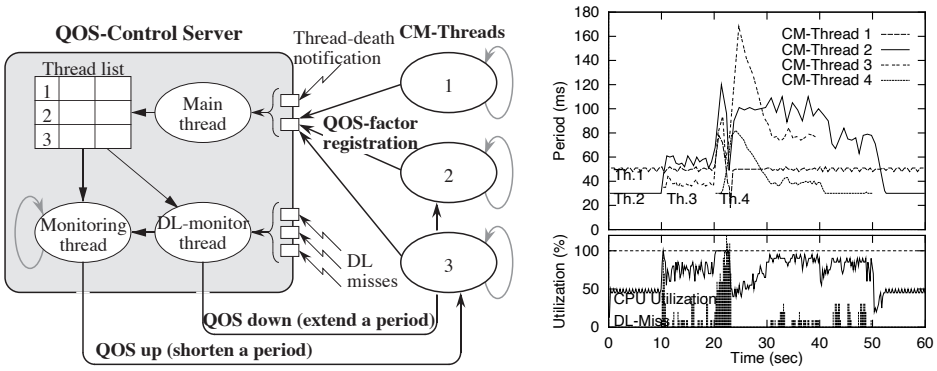

Fig. 1. Experimental QOS-Control Server (a), and the result of QOS control (b)

factor," which describes the characteristics of the session. In accordance with the QOS factors, the server changes the periods of the CM-Threads. To judge the system load, the server monitors the deadline ports of the CM-Threads.

First experiment was carried out with this server using four dummy CM-Threads.[4] Thread 1 represents a session whose period (QOS) is fixed to 50 msec. Thread 2 allows a change in the QOS, but requests 100 msec period in the worst case. Threads 3 and 4 are high-priority sessions started during thread 2. Figure 1(b) shows the actual period of each thread and system status. In this result, QOS control is partially achieved, but there are two major problems: first, the period of each thread jumps up and down before settling, as shown in the section between 20 sec and 30 sec; and second, it vibrates even after the jump is settled.

3 OS Support for Efficient QOS Control

The main reason for the problems is that the OS support is insufficient for efficient QOS control. For more efficient QOS control, we made several extensions to RT-Mach and the QOS-Control Server.

3.1 Extending the Real-Time Thread Model

The largest problem in the experiment is the jump in the period. We found that the problem is caused by the "CATCH UP" semantics of the periodic thread. Figure 2 shows the invocation timing and the actual execution of a periodic real-time thread during overload (hatched area) in RT-Mach. The behavior of the CATCH UP mode is shown in Fig. 2(a). In this mode, the invocation timing is decided in advance and is not changed. If a processing unit is not finished before the next invocation time, the "debt" is transferred to the subsequent invocations.

[4] IBM ThinkPad 755C (9545-L, IntelDX4-75MHz) with RT-Mach MK83g was used.

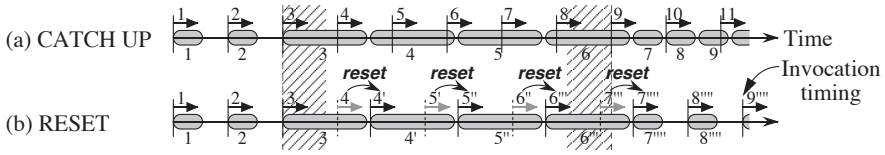

Fig. 2. Behavior of periodic real-time threads during overload

So, even after the overload, the deadlines continue to be missed until the thread catches up its original timing. This causes the jump in the experiment.

In continuous-media processing with QOS control, the settlement of transient overload is more important than the exact invocation timing. Therefore, several new semantics are added to RT-Mach, including a RESET mode, shown in Fig. 2(b). In this mode, the invocation timing is "reset" if a processing unit is not finished before the next invocation. By using the RESET mode, we can create a thread suitable for continuous-media processing. This thread works as a periodic thread if the system load is light, and in best-effort mode in a (transient) overload situation that needs QOS control.

3.2 QOS Control Based on CPU Reservation

By using the RESET mode, the jump-up problem is solved, but there still remains the second "vibration" problem. In the experimental server, it is impossible to avoid deadline misses. CPU utilization at times reaches 100%, and the real-time scheduler in RT-Mach cannot schedule CM-Threads properly. To solve the problem, tighter cooperation between the server and OS is necessary. For this purpose, RT-Mach's CPU reservation mechanism [7] can be used. This mechanism provides a new abstraction named "reserve." Every thread in the system is associated with a reserve, and can gain preferential use of the CPU through it.

Figure 3(a) shows the structure of the new QOS-Control Server based on CPU reservation. Instead of changing the period of CM-Threads directly, the new server "issues" a reserve. The CPU-resource assignment to these reserves is calculated in accordance with the QOS factors. Each CM-Thread adjusts its own QOS (period) to meet the restriction imposed by its reserve. The CPU-time information in the reserve can be used as a hint in making this adjustment. Second experiment was carried out with this new QOS-Control Server. The RESET mode was used for the four dummy CM-Threads. As shown in Fig. 3(b), the behavior of CM-Threads becomes very stable. CPU utilization is also stable at 80%–90%, and the number of deadline misses is almost zero.

In accordance with these experiments, we are now designing a new QOS-control model based on "QOS-Ticket" [8]. In this QOS-Ticket model, QOS control is achieved through cooperation between a QOS Manager and each continuous-media session. The QOS-Ticket, which represents the resource reservation for each session, mediates these two activities. The new QOS-Control Server based on CPU reservation can be considered as a prototype of this model.

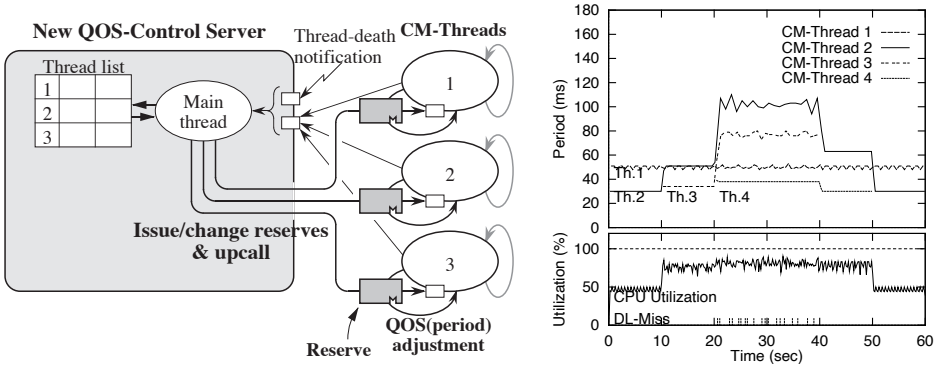

Fig. 3. New QOS-Control Server (a), and the result of QOS control (b)

4 Conclusion

We have described several QOS-control experiments with QOS-Control Servers on RT-Mach. Through these experiments, we found that appropriate OS support is important for effective QOS control. For example, the thread-model extension makes it possible to create a thread suitable for continuous-media processing, and the CPU-reservation mechanism gives more efficient QOS control. Based on these experiments, we are now designing a "QOS-Ticket" model as a new QOS-control scheme.

References

1. H. Tokuda et al.: "Real-Time Mach: Towards a Predictable Real-Time System," *Proc. USENIX Mach Workshop*, pp. 73–82 (1990).
2. K. Kawachiya et al.: "Extending Real-Time Mach for Continuous Media Applications," *Collected Abstracts 4th NOSSDAV*, pp. 55–58 (1993).
3. N. Nishio et al.: "Conductor-Performer: A Middle Ware Architecture for Continuous Media Applications," *Proc. 1st Intl. Workshop on Real-Time Computing Systems and Applications*, pp. 122–131 (1994).
4. H. Tokuda and T. Kitayama: "Dynamic QOS Control based on Real-Time Threads," *Proc. 4th NOSSDAV*, pp. 113–122 (1993).
5. A. Campbell et al.: "A Multimedia Enhanced Transport Service in a Quality of Service Architecture," *Proc. 4th NOSSDAV*, pp. 123–136 (1993).
6. L. Delgrossi et al.: "Media Scaling for Audiovisual Communication with the Heidelberg Transport System," *Proc. ACM Multimedia '93*, pp. 99–104 (1993).
7. C. W. Mercer et al.: "Processor Capacity Reserves: Operating System Support for Multimedia Applications," *Proc. Intl. Conf. on Multimedia Computing and Systems*, pp. 90–99 (1994).
8. K. Kawachiya et al.: "QOS Control of Continuous Media: Architecture and System Support," *IBM Research Report, RT0108*, IBM (1995).

This article was processed using the LaTeX macro package with LLNCS style